|  |  |
| --- | --- |
| Picture 1 | **МИНОБРНАУКИ РОССИИ**  **федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  «Санкт-Петербургский государственный морской технический университет» (СПбГМТУ) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет цифровых промышленных технологий

Направление подготовки 09.03.01.03 "Интеллектуальные технологии киберфизических систем"

**«Лабораторная работа №2»**

Студент 1 курса группы 20121

Очного отделения

Трапезников А. Н.

Проверил:

Поделенюк П. П.

2023

# Цель работы

# Целью данной лабораторной работы является разработка и оптимизация алгоритма для размещения дополнительных фигур на шахматной доске таким образом, чтобы никакая фигура не находилась под боем другой.

# Ход работы

1. Чтение входных данных:

Из файла "input.txt" считываются три числа: N, L, K.

Далее считываются K пар координат уже размещенных фигур.

1. Инициализация доски и размещение известных фигур:

Создается квадратная шахматная доска размером N x N с помощью функции matrx\_builder.

Известные фигуры размещаются на доске с помощью функции create\_board.

1. Была реализована функция piece\_moves, которая принимает координаты фигуры и выводит множество с координатами, куда эта фигура может походить.
2. Была реализована функция posing\_the\_figure, которая на матрице показывает фигуру и клетки, куда эта фигура может походить.
3. Оптимизированный поиск возможных решений:

Используется рекурсивная функция recursion\_for\_all\_arrangements, которая проверяет все возможные комбинации размещения L фигур на доске.

1. Запись результатов:

Найденные решения записываются в файл "output.txt" в виде пар координат.

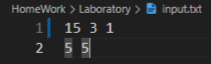
1. Вывод доски и результатов на консоль:

Выводится количество найденных решений и доска, где фигуры обозначены символом "#", ходы - "\*", а пустые клетки - "0".

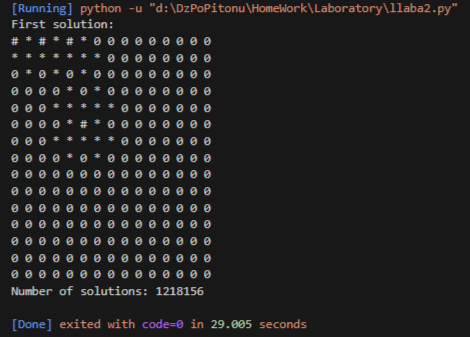
# Результат работы

# Когда решения есть

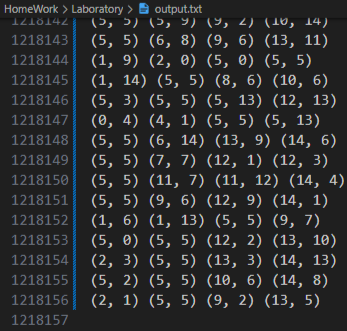
Input.txt



Консоль

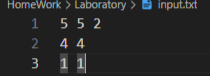


Часть файла Output.txt

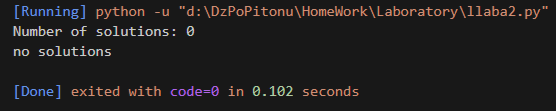


# Когда решений нет

Input.txt



Консоль



Output.txt не создан

# Вывод

В ходе выполнения лабораторной работы был разработан и оптимизирован алгоритм размещения дополнительных фигур на шахматной доске. Полученный алгоритм работает эффективно и находит все возможные решения задачи.

Были проведены тесты с различными входными данными, включая доски размером до 20х20. Программа успешно справляется с поставленной задачей, обеспечивая оптимальное распределение фигур и выводя результаты в файл "output.txt". Количество найденных решений выводится на консоль для удобства анализа.

Таким образом, цель работы достигнута, и разработанный алгоритм может быть использован для эффективного решения аналогичных задач размещения фигур на шахматной доске.

# Листинг кода

def matrx\_builder(N: int) -> list[list[str]]:

    return [['0' for \_ in range(N)] for \_ in range(N)]

def piece\_moves(x, y):

    moves = {

        (x - 1, y - 1), (x - 1, y),

        (x - 1, y + 1), (x, y - 1),

        (x, y + 1), (x + 1, y - 1),

        (x + 1, y), (x + 1, y + 1),

        (x - 2, y - 1), (x - 2, y + 1),

        (x - 1, y - 2), (x - 1, y + 2),

        (x + 1, y - 2), (x + 1, y + 2),

        (x + 2, y - 1), (x + 2, y + 1)

    }

    return moves

def posing\_the\_figure(x: int, y: int, matrix: list[list[str]]) -> list[list[str]]:

    dragon\_moves = [

        (x - 1, y - 1), (x - 1, y),

        (x - 1, y + 1), (x, y - 1),

        (x, y + 1), (x + 1, y - 1),

        (x + 1, y), (x + 1, y + 1),

        (x - 2, y - 1), (x - 2, y + 1),

        (x - 1, y - 2), (x - 1, y + 2),

        (x + 1, y - 2), (x + 1, y + 2),

        (x + 2, y - 1), (x + 2, y + 1)

    ]

    matrix[x][y] = '#'

    for i in dragon\_moves:

        m, n = i[0], i[1]

        if 0 <= m < len(matrix) and 0 <= n < len(matrix):

            matrix[m][n] = '\*'

    return matrix

def create\_board(matrix: list[list[str]], posed\_figures: list[tuple[int, int]]) -> list[list[str]]:

    for x, y in posed\_figures:

        posing\_the\_figure(x, y, matrix)

    return matrix

def print\_board(matrix: list[list[str]]):

    for row in matrix:

        print(" ".join(row))

def recursion\_for\_all\_arrangements(N: int, L: int, solutions: set[tuple[int, int]], solution: list[tuple[int, int]], cnt: int):

    if cnt == L:

        unique\_solution = tuple(sorted(solution))

        solutions.add(unique\_solution)

        # Вывод первого решения

        if len(solutions) == 1:

            print("First solution:")

            print\_board(create\_board(matrx\_builder(N), unique\_solution))

        return

    for i in range(N):

        for j in range(N):

            if (i, j) not in solution and not piece\_moves(i, j).intersection(solution):

                solution.append((i, j))

                recursion\_for\_all\_arrangements(N, L, solutions, solution, cnt + 1)

                solution.pop()

if \_\_name\_\_ == "\_\_main\_\_":

    file = open("D:/DzPoPitonu/HomeWork/Laboratory/input.txt", "r")

    N, L, K = map(int, file.readline().split())

    posed\_figures = []

    solutions = set()

    for line in file.readlines():

        x, y = map(int, line.split())

        posed\_figures.append((x, y))

    file.close()

    recursion\_for\_all\_arrangements(N, L, solutions, posed\_figures, 0)

    print(f"Number of solutions: {len(solutions)}")

    if solutions:

        solutions\_str = [" ".join(map(str, solution)) + "\n" for solution in solutions]

        with open("D:/DzPoPitonu/HomeWork/Laboratory/output.txt", "w") as output\_file:

            output\_file.writelines(solutions\_str)

    else:

        print('no solutions')